НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Програмне забезпечення інформаційно-комунікаційних систем»
спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Автоматизована система пошуку медіафайлів за визначеними класами»

Виконав:
студент IV курсу, групи ІТ-61
Інамов Сергій Валерійович _____

Керівник:
Асистент кафедри АУТС
Шинкевич Микола Костянтинович _____

Рецензент:
Доцент кафедри ОТ, кандидат технічних наук
Верба Олександр Андрійович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення інформаційно-комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Інамову Сергію Валерійовичу

1. Тема проєкту «Автоматизована система пошуку медіафайлів за визначеними класами», керівник проєкту асистент Шинкевич Микола Констянтинович, затверджені наказом по університету від 07 травня 2020 р. №1081-с

2. Термін подання студентом проєкту 09.06.2020

3. Вихідні дані до проєкту

Мови програмування JavaScript, Scala, Python, бібліотека JavaScript Puppeteer, фреймворк Scala Akka, бібліотека Python PyTorch, середовище програмування IntelliJ Idea, СУБД – PostgresSQL, брокер повідомлень Apache Kafka.

4. Зміст пояснювальної записки

1. Вступ 2. Опис предметної області 3. Аналіз існуючих рішень 4. Аналіз вимог до програмного забезпечення 5. Вибір технологій розробки 6. Розробка структури системи 7. Тестування програмного забезпечення 8. Впровадження та використання розробленої системи

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо)

Діаграма варіантів використання, діаграма компонентів, діаграма послідовностей, діаграма відношень сутностей у базі даних

6. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Вибір тематичного напрямку та узгодження теми дипломного проекту	22.02.2020	
2	Аналіз теоретичних матеріалів та вивчення предметної області	15.04.2020	
3	Розробка технічного завдання, вибір методів та засобів реалізації задачі	24.04.2020	
4	Огляд існуючих рішень з тематики роботи	27.04.2020	
5	Розробка структури прототипу та проектування системи	06.05.2020	
6	Реалізація проекту	20.05.2020	
7	Налагодження та перевірка програми	23.05.2020	
8	Оформлення пояснювальної записки	03.06.2020	
9	Передзахист дипломного проекту	04.06.2020	

Студент Сергій ІНАМОВ

Керівник Микола ШИНКЕВИЧ

АНОТАЦІЯ

Інамов С.В. Автоматизована система пошуку медіафайлів за визначеними класами. КПІ ім. Ігоря Сікорського, Київ, 2020.

Ключові слова: Scala, Javascript, бібліотека PyTorch, бінарні нейронні мережі, діаграма варіантів використання, бібліотека Puppeteer, парсинг, RESTFul API, реактивна архітектура.

Основна частина документу викладена у пояснювальній записці, виконаній на 61 сторінці, та містить 5 рисунків та 24 таблиці.

Об’єктом розробки є система пошуку медіафайлів.

Мета розробки – створення система пошуку медіафайлів за визначеними класами.

У дипломному проекті проведено аналіз існуючих рішень та на базі результатів розроблено система пошуку зображень, критерії пошуку в якої визначаються класами зображень. Система реалізована у вигляді мікросервісної архітектури, та має велику кількість тестів, які покривають основні сценарії її використання.

Отримані результати можуть бути корисними при створенні аналогічних чи подібних систем.

SUMMARY

Inamov S. V. Automated image search system by predefined classes. Igor Sikorsky KPI, Kyiv, 2020.

Keywords: Scala, Javascript, library PyTorch, binary neural network, use case diagram, library Puppeteer, parsing, RESTFul API, reactive architecture.

The bulk of the document is outlined in the explanatory note, with 61 pages, and contains 5 figures and 24 tables.

The object of development is image search system.

The purpose of the development – creation of image search system by predefined classes.

Пояснювальна записка
до дипломного проекту
на тему: «Автоматизована система пошуку медіафайлів за визначеними класами»

Київ – 2020 рік

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

СКБД – система керування базою даних;

API (англ. application programming interface) – програмний інтерфейс додатку;

DOM (англ. Document Object Model) – об’єктна модель документа;

FTP (англ. File Transfer Protocol) – протокол передачі файлів;

HTTP (англ. HyperText Transfer Protocol) – протокол передачі гіпертексту;

MIME (Multipurpose Internet Mail Extension, Багатоцільові розширення пошти Інтернету) – специфікація для передачі по мережі файлів різного типу: зображень, музики, текстів, відео, архівів та інше.

REST (англ. Representational State Transfer) – передача стану керування;

UUID (англ. universally unique identifier «универсальный уникальный идентификатор») – це 128-бітове число, яке використовується для ідентифікації інформації в комп'ютерних системах.

ВСТУП

У епоху цифрових технологій та глобалізації даних будь-яке питання стосовно пошуку та структурування інформації є надзвичайно актуальним. І зазвичай мова йде про величезні обсяги даних.

Великою кількістю інформаційного або медіа контенту обладає ледве не кожен сайт чи застосунок. Мільйони терабайт даних щодня розповсюджується мережею Інтернет. Але на більшості ресурсів присутня лише інформація, а не дані. Кардинальна різниця між даними та інформацією полягає у структурованості та можливості логічно обробляти ці дані за допомогою цифрових засобів.

Тривалий час з років зародження інтернету, весь інформаційний простір представляє собою велике звалище гіпертекстової інформації, яка була призначена лише для читання та розуміння людиною. Більшість сервісів навіть і не думали про стандартизоване API. З плином часу, інформаційне суспільство усвідомило необхідність більшої формалізації та структуризації даних. Тобто з'явилася потреба в глобальній семантичній мережі, в онтологічному підході. Так згодом сформувалися світові стандарти представлення даних, такі як наприклад сімейство стандартів на мови опису, що включає XML, XML Schema, RDF, RDF Schema, OWL. [1]

Зі зростанням кількості сайтів у мережі, та спрямованості їх діяльності, однієї формалізації стало замало. Не усі сервіси мають ресурси на реалізацію цих стандартів, а деякі навпаки не бажають розповсюджувати контент у вигляді структурованих даних, а не у вигляді інформації, корисної тільки для одиничних користувачів.

В обхід цим перепонам почали з’являтися перші парсери даних. Програмні реалізації автоматичного збору гіпертекстової інформації з мережі у завчасно зазначеному вигляді та формування структурованих даних у результаті. Є багато сперечань щодо законності такого збору інформації, але окремих законів, які регулюють таку діяльність немає. Найчастіше юристи відштовхуються від

авторського права та конфіденційності інформації. Якщо дані є загальнодоступними, то немає ніяких обмежень на способи їх обробки.

Коли користувач бачимо цікаву статтю або фотоальбом на сайті, він має змогу ознайомитися з матеріалом або вручну скопіювати частину контенту, якщо це дозволено правилами сайту та програмною реалізацією. Поряд з потребами користувача на отримання інформації, можливі такі ж потреби в автоматизованій обробці цієї інформації, яка є загальнодоступною. Це можуть бути агрегатори цінових показників різного роду ринкових товарів, відслідковування спортивних подій чи просто ведення статистики. Подібного роду систему пошуку інформації називають парсерами. По своїй суті, пошукові системи, як-то Google, також можно вважати свого роду парсерами, тому що їх пошукові боти постійно індексують веб-сторінки.

Цікаво те, що більшість з них націлені на збирання текстової інформації. Деякі вміють збирати ще й медіа файли, в основному зображення. Але якщо веб-сторінка не пропонує опис до зображення, то змоги автоматично структурувати зображення відповідно до того, що на них знаходиться, неможливо. Така можливість була б корисною для обробки великої кількості неструктурованих зображень, як-от пошук тренувальних даних для нейронної мережі, пошук даних для статистики або просто пошук певних фігур чи образів.

У даному проєкті буде проведена спроба створити можливість пошуку та структуризації графічних зображень відповідно до визначених класів. Ці зображення будуть завантажуватися із вказаних користувачем веб-сторінок за допомогою парсингу.

Дипломний проєкт складається з наступних розділів: опис предметної області, огляд існуючих рішень, аналіз вимог до програмного забезпечення, вибір технологій розробки, розробка системи, тестування програмного забезпечення, впровадження та використання розробленої системи.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Обґрунтування доцільності розробки

Системи пошуку та структурування інформації завжди знаходили своє середовище застосування. Перш за все це – агрегатори даних. Вони використовують парсери для пошуку інформації з різних джерел, групують її та централізують на своєму ресурсі. Для агрегаторів, які мають справу з зображеннями, так названі, фотостоки, було б ефективно поповнювати свою базу таким чином.

Також розроблювана система може знайти свій застосунок у навчанні різного роду нейронних мереж. Адже для тренування більшості з них необхідний великий масив “навчальних” (тобто структурованих за класами) зображень. Логічно зробити висновок, що таким чином відчутно зменшується витрачений на пошук даних для тренування час.

1.1.1 Опис та аналіз предметного середовища

Автоматизована система пошуку медіафайлів за визначеними класами, що розробляється – це програмне рішення задачі автоматизованого завантаження зображень та пошуку серед них таких, які відповідають заданому класу, доступному у системі.

Зображення несуть не менше корисної інформації, ніж текст, але за умови якщо у відповідність цим зображенням поставлена корисна інформація, як-от опис зображення, джерело отримання зображення, структурування за певною ознакою. Складність пошуку зображень значно більша звичайного пошуку по гіпертекстовим документам. Хоча для спрощення пошуку зображень присутні спеціальні HTML теги та атрибути до них, але за релевантність цієї інформації відповідає виключно автор веб-сторінки. Таким чином, втрачається можливість створювати свої критерії пошуку зображень, перш за все за допомогою власної класифікації.

1.1.2 Аналіз функціональних особливостей системи

Дана система являє собою набір взаємопов’язаних сервісів, та надає програмний інтерфейс взаємодії у вигляді HTTP REST API та підтримку FTP з’єднання для отримання доступу до результатів виконання роботи програми.

Варіанти використання сервісу діляться за наступними напрямками:

1. взаємодія з інтерфейсом керування задачами;
2. взаємодія з інтерфейсом керування класам и зображень;
3. перегляд списку публічно доступних класів.

Перші два пункти доступні тільки для авторизованих користувачів. Реєстрація у системі проводиться з ціллю встановлення способу однозначної ідентифікації користувача у подальших запитах.

Також у системі передбачено створення на кожну нову задачу FTP посилання через яке можна отримати результати пошуку картинок.

1.2 Призначення розробки

Розроблений продукт є програмним інструментом, який може застосовуватися у багатьох сферах діяльності. Зокрема, може стати основним джерелом даних для навчання нейронних мереж, агрегаторів, інформаційних порталів, “фотостоків”, статистичних застосунків, тренування нейронних мереж.

1.3 Цілі та задачі розробки

Основною метою розробки даного продукту є створення можливості гнучкого пошуку зображень за власно визначеними класами. Можливість автоматизації цього пошуку в поєднані із зручним програмним інтерфейсом. Створення можливості для постійного масштабування такої системи для розширення доступних класів пошуку.

1.4 Висновок до розділу

Розробка даного програмного продукту потребує детального вивчення доступних технологій для реалізації, загальної структури гіпертекстових документів, існуючих можливостей автоматизації збору інформації з мережі, а також технологій розпізнавання зображень за прийнятний час. Відповідь на ці питання дасть змогу розробити первинну реалізацію описаного продукту та забезпечити користувача базовою функціональністю.

Авжеж, необхідно проаналізувати існуючі рішення та виділити критичні моменти у їх реалізації, переваги та недоліки. Це необхідно для отримання більш широкого розуміння предметної області, наявних труднощів, важливих моментів реалізації та розробки корисного у певній сфері застосування програмного продукту.

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

2.1 Існуючі системи пошуку медіафайлів за визначеними класами

Далі будуть розглянуті деякі існуючі рішення систем інших розробників.

2.1.1 Google Images

Google Images – це спеціальний сервіс Google для пошуку картинок в Інтернеті. В основі роботи сервісу лежить Googlebot–Image – пошуковий робот, що сканує сторінки для індексу картинок та робить пошук зображень відповідно до деяких правил. Цей робот шукає відповідні теги, що репрезентують зображення (img тег), дістають звідти посилання на джерело картинки та опис. Опис зазвичай береться із кількох спеціальних атрибутів цього тегу. [2]

Google Images має функцію пошуку по зображенню для виконання зворотного пошуку зображень. На відміну від традиційного пошуку зображень, ця функція позбавляє від необхідності вводити ключові слова і терміни в поле пошуку Google. Замість цього користувачі виконують пошук, відправляючи зображення в якості запиту. Результати можуть включати в себе схожі зображення, веб-результати, сторінки з зображенням і різної розподільної якості зображення.

Точність результатів пошуку по зображенню вище, якщо зображення для пошуку більш популярно. Крім того, Google Search by Image запропонує «краще припущення для цього зображення» на основі описових метаданих результатів.

Основні переваги цього сервісу:

1. швидкодія та надійність. Це зумовлено тим, що використовуються передові технології розробки, повна відповідність реалізації концептуальним засадам формування веб-сторінок;
2. великі обчислювальні можливості компанії-розробника;
3. одна з найбільших доступних баз зображень.

Google images має дуже широкі функціональні можливості. Окрім звичайного пошуку по ключовим словам, є можливість задавати розширений

пошук за розміром, за кольором, за типом, по формі картинок, по часу, по формату файлів и т. д. Також є можливість пошуку по зображенню, котра надає змогу шукати схожі картинки у відповідності до наданого зображення.

Таким чином, у даного сервіса присутня схожа до розроблюванного продукту функціональність. Але є певні ключові відмінності, як-от:

1. неможливо знайти картинку, яка не була проіндексована;
2. неможливо знайти картинку опис якої відрізняється від реального зображення;
3. неможливо створити свій клас для пошуку, якщо такого немає у системі.

Ці недоліки зумовлені концепцією пошуку, якої дотримується даний сервіс. Розроблюваний продукт націлений саме на цю відсутню функціональність.

2.1.2 Depositphotos

Depositphotos – це фотостоковое агентство з власним інтернет-фотобанком, яке займається продажем різноманітних зображень, об'єднаних в десятки категорій, на умовах ліцензії royalty-free. Фотобанк – це сервіс зі збору, зберігання і презентації зображень або інших медійних матеріалів, з метою посередництва між авторами та потенційними покупцями. Фотобанк бере на себе обов'язки по оформленню каталогів робіт, залученню покупців, супроводу угод, прийому оплати від покупців і виплату винагороди авторам.

В основі сервісу – веб сайт який розділяє користувачів на продавців та покупців. Перші мають змогу шукати зображення за різноманітними критеріями, додавати їх у свій кошик та оплачувати придбані картинки. Останні – джерело контенту на сайті, вони проходять сертифікацію та отримають змогу отримувати прибуток за продаж своїх картинок. [3]

До ключових переваг Depositphotos можна віднести:

1. високу якість зображень;
2. їх ліцензійність;

3. гнучкий пошук;

4. гнучка фільтрація контенту.

Цей перелік зумовлений наявною жосткою сертифікацією постачальників.

До недоліків Depositphotos слід віднести:

1. відносно висока ціна зображень, що унеможливлює їх використання у великих масштабах, як-от тренування нейромережі;

2. зображення несуть собою цінність тільки як медіафайли і не несуть ніякої інформаційної цінності;

3. самі зображення завчасно класифіковані, що певним чином звужує гнучкість пошуку;

4. порівняно невелика база зображень.

2.1.3 TinEye

TinEye – це система зворотного пошуку зображень, розроблена і запропонована компанією Idée, Inc. Це перша пошукова система зображень в Інтернеті, яка використовує технологію ідентифікації зображень, а не ключові слова, метадані або водяні знаки. TinEye дозволяє користувачам здійснювати пошук не по ключовим словами, а по зображеннях. Після відправки зображення TinEye створює «унікальну і компактну цифрову підпис або відбиток пальця» зображення і зіставляє його з іншими проіндексованими зображеннями. Ця процедура може відповідати навіть сильно відредагованим версіями представленого зображення, але зазвичай не повертає схожі зображення в результатах.

Система працює наступним чином. Користувач завантажує зображення в пошукову систему або надає URL-адресу зображення або сторінки, що містить зображення. Пошукова система буде шукати інше використання зображення в Інтернеті, в тому числі відредаговані фотографії на основі цього зображення, і повідомляти дату і час, коли вони були опубліковані. TinEye не розпізнає контури об'єктів і не виконує розпізнавання осіб, але розпізнає всі зображення і деякі змінені

версії цього зображення. Це включає в себе менші, великі і обрізані версії зображення.

TinEye здатний шукати зображення в форматі JPEG, GIF або PNG. З 2009 року інші формати, що містять зображення в Інтернеті, такі як Adobe Flash, недоступні для пошуку.

До основних переваг TinEye слід віднести:

2. пошук зображень не за ключовими словами, а за допомогою ідентифікації зображуваних елементів;
3. швидкодія.

До недоліків TinEye слід віднести:

4. зображення стають доступні для пошуку тільки після їх індексації пошуковою системою;
5. користувач отримує тільки «схожі» зображення, але не може визначити конкретний клас зображень;
6. порівняно з іншими пошуковими системами, невелика база проіндексованих зображень.

2.3 Висновок до розділу

Аналіз розглянутих рішень показав, що їх спільним недоліком є відсутність пошуку за власно визначеними класами. Також відсутня можливість автоматизованого збору зображень. Саме цю функціональність пропонує сервіс що розробляється. Водночас, відсутність схожих реалізацій ускладнює процес проєктування, так як немає джерела досвіду, щоб врахувати можливі труднощі розробки подібної системи.

3 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Функціональні вимоги до системи

До системи наявні наступні функціональні вимоги:

1. взаємодія з інтерфейсом керування задачами:
    1. створення задачі;
    2. перевірка стану задачі;
    3. відображення історії задач;
    4. відміна виконання задачі.
2. взаємодія з інтерфейсом керування класами зображень:
    1. додавання власного класу у систему;
    2. редагування налаштувань доступу до класу (публічний/приватний);
    3. видалення власного класу з системи;
    4. перегляд списку власних класів.
3. перегляд списку публічно доступних класів;
4. авторизація.

3.2 Сценарії використання системи

Детальний огляд усіх варіантів використання системи відображений на діаграмі варіантів використання (прецедентів) на кресленику ІТ61.080БАК.004 Д1.

Таблиця 3.1 – Варіанти використання системи

Шифр варіанту використання	Назва варіанту використання
UC-1	Перегляд усіх класів системи.
UC-2	Запит на авторизацію.
UC-2.1	Перевірка статусу задачі.

UC-2.2	Створити задачу.

Продовження таблиці 3.1

Шифр варіанту використання	Назва варіанту використання
UC-2.3	Відмінити виконання задачі.
UC-2.4	Додати свій клас.
UC-2.5	Видалити свій клас.
UC-2.6	Переглянути список своїх класів.

У таблицях 3.2-3.15 приведені описи кожного з варіантів використання. Опис варіанту використання з шифром UC-1 зазначено у таблиці 3.2.

Таблиця 3.2 – Варіант використання UC-1

Назва	Перегляд усіх класів системи.
Опис	Авторизований або неавторизований користувач має змогу переглядати список усіх публічних класів системи.
Учасники	Користувач.
Передумови	
Постумови	Відображено список усіх публічних класів.
Основний сценарій	Користувач робить HTTP запит до системи по шляхом /public/classes; Система відображає список усіх класів.

Опис варіанту використання з шифром UC-2 зазначено у таблиці 3.3.

Таблиця 3.3 – Варіант використання UC-2

Назва	Авторизація
Опис	Користувач має змогу авторизуватися у системі.

Учасники	Користувач.
Передумови	У системі немає користувача з такою самою email адресою.

Продовження таблиці 3.3

Постумови	Користувач отримав ключ–токен для доступу до окремих частин системи.
Основний сценарій	1) Користувач робить HTTP запит до системи по шляхом /auth. 2) Система реєструє користувача.
Розширення	Система відповідає на некоректно введені дані помилкою.

Опис варіанту використання з шифром UC-2.1 зазначено у таблиці 3.4.

Таблиця 3.4 – Варіант використання UC-2.1

Назва	Перевірка статусу задачі.
Опис	Користувач перевіряє статус виконання задачі відповідно до її ідентифікатора.
Учасники	Користувач.
Передумови	Користувач аутентифікований у системі.
Постумови	Користувач отримує статус задачі за вказаним ідентифікатором.
Основний сценарій	1) Користувач робить HTTP запит до системи по шляхом /tasks/{id}; 2) Система повертає актуальний на момент запиту статус задачі за певним ідентифікатором.

Опис варіанту використання з шифром UC-2.2 зазначено у таблиці 3.5.

Таблиця 3.5 – Варіант використання UC-2.2

Назва	Перевірка статусу задачі.

Опис	Користувач створює задачу на пошук зображень за вказаними посиланнями та відповідно до вказаного класу.

Продовження таблиці 3.5

Учасники	Користувач.
Передумови	Користувач аутентифікований у системі.
Постумови	Користувач отримує ідентифікатор створеної задачі.
Основний сценарій	1) Користувач робить HTTP запит до системи по шляхом /tasks/{id}; 2) Система створює нову задачу на пошук зображень та починає її виконувати відповідно до наявних ресурсів.

Опис варіанту використання з шифром UC-2.3 зазначено у таблиці 3.6.

Таблиця 3.6 – Варіант використання UC-2.3

Назва	Відміна виконання задачі.
Опис	Користувач відміняє виконання раніше запущеної задачі.
Учасники	Користувач.
Передумови	користувач аутентифікований у системі. у системі існує задача із запитуємим ідентифікатором, що відповідає користувачу.
Постумови	Користувач отримує підтвердження відміни задачі.
Основний сценарій	1) Користувач робить HTTP запит до системи по шляхом /tasks/{id}; 2) Система відміняє виконання задачі за вказаним ідентифікатором.

Опис варіанту використання з шифром UC-2.4 зазначено у таблиці 3.7.

Таблиця 3.7 – Варіант використання UC-2.4

Назва	Додавання користувацького класу.
Опис	Користувач додає у систему власний клас у вигляді нетренованої нейронної моделі.

Продовження таблиці 3.7

Учасники	Користувач.
Передумови	Користувач аутентифікований у системі.
Постумови	Користувач отримує підтвердження додавання класу від системи.
Основний сценарій	1) Користувач робить HTTP запит до системи по шляхом /classes. 2) Додається новий клас до системи відповідно до налаштувань користувача.

Опис варіанту використання з шифром UC-2.5 зазначено у таблиці 3.8.

Таблиця 3.8 – Варіант використання UC-2.5

Назва	Видалення користувацького класу.
Опис	Користувач видаляє з системи власний клас за вказаним ім’ям.
Учасники	Користувач.
Передумови	. користувач аутентифікований у системі. . у системі існує клас із запитуємим ім’ям, що відповідає користувачу.
Постумови	Користувач отримує підтвердження видалення класу з системи.
Основний сценарій	1) Користувач робить HTTP запит до системи по шляхом /classes/{id}; 2) Система видаляє користувацький клас.

Опис варіанту використання з шифром UC-2.6 зазначено у таблиці 3.9.

Таблиця 3.9 – Варіант використання UC-2.6

Назва	Перегляд списку класів користувача.

Продовження таблиці 3.9

Опис	Система повертає список класів, створених користувачем.
Учасники	Користувач.
Передумови	Користувач аутентифікований у системі.
Постумови	Користувач отримує список своїх класів, доданих раніше у систему.
Основний сценарій	1) Користувач робить HTTP запит до системи по шляхом /classes; 2) Система повертає список класів користувача.

3.3 Нефункціональні вимоги до системи

Проаналізувавши предметну область та провівши огляд існуючих рішень, можна скласти наступні нефункціональні вимоги до програмного забезпечення.

Автоматизована система пошуку медіафайлів за визначеними класами, що розробляється, повинна мати здатність легко інтегруватися з іншими програмними системами. Така необхідність випливає із широкого спектру можливостей застосування продукту. Виходячи з цього необхідно мати чіткий програмний інтерфейс на основі зручного та розповсюдженого серед багатьох систем протоколу обміну даними. Далі приведені ключові аспекти, необхідні для реалізації описаних нефункціональних вимог:

1. масштабованість – система має бути достатньо гнучкою для того, щоб у майбутньому була можливість розширювати функціонал за допомогою створення додаткових модулів;
2. слабка зв’язність компонентів – оскільки обрана мікросервісна архітектура, компоненти системи повинні бути легко замінні та легко змінювані;

3. оскільки виконання задач виконується впродовж деякого часу, то необхідно мати можливість продовжити їх виконання після, наприклад, перезапуску програми;

4. повторне використання компонентів – компоненти системи мають бути розроблені таким чином, щоб у майбутньому була можливість їх повторного використання. Тобто система частково або повністю повинна складатися з частин, написаних раніше компонентів або частин іншої системи. Це значно скоротить трудовитрат при розробці складної систем;

5. безпека – оскільки система є високонавантаженою, то необхідно забезпечити стійкість до різного типу атак, зокрема DDoS.

3.4 Висновки до розділу

У даному розділі були описані функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення. Розроблена діаграма прецедентів, яка відображає сутність функціональних вимог та структуру їх використання користувачем. Висунуті нефункціональні вимоги походять з досвіду розроби програмного забезпечення та підкреслюють загальноприйняті вимоги до програмного забезпечення.

4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

Розробляючи складні програмні системи, важливо детально дослідити доступні технології для реалізації поставлених технічних завдань.

Технології розробки програмних продуктів налічують широкий спектр підходів та реалізацій. Існує ряд вже створених інструментів, які пришвидшують розробку, передусім це бібліотеки, фреймворки, архітектурні шаблони, окремі сервіси, що гнучко налаштовуються під цілий ряд потреб. Бібліотеки та фреймворки розроблені для певної мови програмування або мають декілька реалізацій під різні мови. З іншого боку, окремі сервіси, що можна гнучко налаштовувати, виконують обслуговуючі функції та від мови не залежать. Далі буде детально розглянуто конкретні технології, що застосовувалися у розробці проєкту.

Дана система має велику кількість різнофункціональних компонентів, частина з яких вимагає специфічних фізичних ресурсів, що дозволяють витримувати великі навантаження, як обчислювальні, так і мережеві. Зазвичай, такі рішення будуються на основі мікросервісної архітектури, що дозволяє розподілити навантаження у відповідності до потреб кожного компонента такої архітектури. Основні принципи такого підходу:

1. розподіл компонентів системи по функціональним ознакам;
2. чіткий протокол взаємодії між сервісами;
3. гнучкість та масштабованість системи за рахунок її модульності;
4. для сервісів, що тримають стан програми у пам'яті: добре спроєктоване розподілення спільних ресурсів, безпечного з точки зору конкурентного доступу до даних.

Далі будуть розглянуті технології розробки, виходячи з описаних функціональних та нефункціональних, особливостей предметної області та проаналізованих існуючих рішень.

Для забезпечення легкої інтеграції з іншими системами, було обрано архітектурний підхід “REST API” на базі протоколу HTTP. REST - це скорочення

від REpresentational State Transfer. Це архітектурний стиль для розподілених гіпермедіа систем. Як і будь-який інший архітектурний стиль, REST також має свої шість напрямних обмежень, які повинні бути виконані, якщо інтерфейс повинен називатися RESTful. Ці принципи перераховані нижче. [4]

1. клієнт-сервер – відокремлюючи проблеми користувальницького інтерфейсу від проблем зберігання даних, покращується переносимість призначеного для користувача інтерфейсу на кілька платформ і покращуємо масштабованість за рахунок спрощення компонентів сервера;

2. без збереження стану – запити від клієнта до сервера повинен містити всю інформацію, необхідну для розуміння запиту, і не може використовувати будь-якої збережений контекст на сервері. Тому стан сеансу повністю зберігається на клієнті;

3. кешованість – це обмеження вимагають, щоб дані у відповіді на запит були явно або неявно позначені як кешовані або не кешовані. Якщо відповідь кешується, клієнтському кешу надається право повторно використовувати ці дані відповіді для наступних еквівалентних запитів;

4. уніфікований інтерфейс – завдяки застосуванню принципу спільності розробки програмного забезпечення до інтерфейсу компонента спрощується загальна архітектура системи і поліпшується видимість взаємодій. Щоб отримати уніфікований інтерфейс, необхідно кілька архітектурних обмежень для управління поведінкою компонентів. REST визначається чотирма інтерфейсними обмеженнями: ідентифікація ресурсів; маніпулювання ресурсами через подання; інформативні повідомлення; і гіпермедіа як двигун стану програми;

5. багаторівневість системи. Стиль багаторівневої системи дозволяє архітектурі складатися з ієрархічних шарів, обмежуючи поведінку компонента таким чином, що кожен компонент не може «бачити» за межами безпосереднього рівня, з яким вони взаємодіють;

6. код за запитом (необов'язково) – REST дозволяє розширювати функціональність клієнта шляхом завантаження та виконання коду в формі аплетів

або скриптів. Це спрощує роботу клієнтів за рахунок зменшення кількості функцій, необхідних для попередньої реалізації.

Мови програмування було обрано відповідно до потреб кожного окремого сервісу систему.

Для реалізації API шлюзу та сервіса для завантаження зображень було використано мову програмування Scala у поєднанні з фреймворками Akka та Slick. Таке рішення прийнято на основі особливостей цих цього стеку технологій. Перш за все – це орієнтованість на мікросервісну реактивну архітектуру та висока швидкість розробки та доставки до експлуатації.

Для реалізації сервісу, що співставляє зображення з класами, було обрано мову програмування Python та бібліотеку для роботи з нейронними мережами – Pytorch. Цей вибір обумовлено простотою написання коду на мові Python, що дозволяє зосередитись на описі логіки функціонування обробки картинок нейронними мережами. Pytorch був обраний за широкий функціонал можливостей, гнучкі інструменти навчання та використання нейронних мереж, а також за відносну простоту використання.

Для пошуку та збору картинок з веб-сайтів було проведено дослідження існуючих технологій, та обрано найбільш сучасну бібліотеку для керування програмним інтерфейсом браузера – Puppeteer. Дана бібліотека надає можливість програмно керувати сучасним браузером Chromium у різних варіантах використання, зокрема у “headless” режимі. Відштовхуючись від цього вибору, для цього сервісу також було обрано мову програмування JavaScript та програмну платформу Node.js.

Для міжсервісної взаємодії був обран брокер повідомлень Apache Kafka. Використання архітектур, яка базується на обміну повідомленнями, що є абстракціями подій у системі, надає нам змогу побудувати реактивну архітектуру системи [5]. Основні маніфести реактивного програмування зображені на рисунку 4.

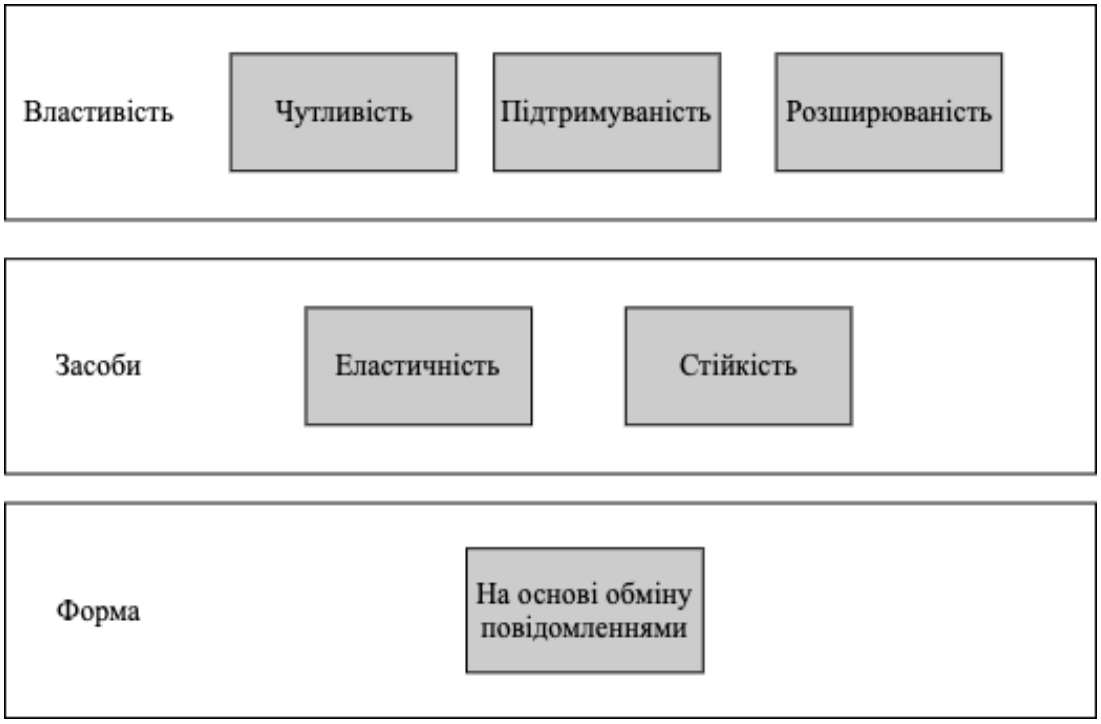

Рисунок 4.1 – структура реактивних властивостей.

Далі будуть розглянуті основні маніфести реактивного програмування:

7. чутливість. Система відповідає своєчасно, якщо це взагалі можливо. Чутливість є наріжним каменем зручного і корисного додатка, але, крім цього, вона дозволяє швидко виявляти проблеми і ефективно їх усувати. Чуйні системи орієнтовані на забезпечення швидкого та узгодженого часу відгуку, встановлюючи надійні верхня межа, щоб забезпечити стабільну якість обслуговування;

8. розширюваність. Додаток може бути розширено до необхідних масштабів. Це досягається за рахунок надання додатком еластичності, властивості, яке дозволяє системі розтягуватися або стискатися (додавати або прибирати вузли) на вимогу. Крім того, така архітектура робить можливим розширюватися або скорочуватися (розвертатися на більшій або меншій кількості процесорів) без необхідності перепроєктування або переписування програми;

9. підтримуваність. За рахунок модульності, система може розроблятися одночасно багатьма спеціалістами. Також в подальшому це полегшить зміну коду окремих компонентів за рахунок більш зрозумілого співставлення сервіс – виконувана функція.

10. гнучкість. Система залишається чуйною під різними навантаженнями. Реактивні Системи здатні реагувати на коливання в швидкості вхідних потоків, збільшуючи або зменшуючи кількість виділених на їх обслуговування ресурсів. Для цього архітектура не повинна допускати наявності централізованих вузьких місць або конкуренції за ресурси, що дозволяє сегментувати або реплікувати компоненти, розподіляючи між ними вхідні дані.

11. стійкість. Система залишається доступною навіть в разі відмов. Це відноситься не тільки до високодоступних, критично важливих додатків - без стійкості будь-яка система при збої втрачає чуйність. Стійкість досягається за рахунок реплікації, стримування, ізоляції та делегування. Ефект від відмов удержівается всередині компонентів, ізолюючи їх один від одного, що дозволяє їм виходити з ладу і відновлюватися, не порушуючи роботу системи в цілому.

12. на основі обміну повідомленнями: Реактивні системи використовують асинхронний обмін повідомленнями, щоб встановити межі між компонентами і забезпечити слабку зв'язаність, ізоляцію і прозорість розміщення. Відкритий обмін повідомленнями робить можливими регулювання навантаження, гнучкість і управління потоком, для чого в системі створюються і відслідковуються черги повідомлень і в разі необхідності використовується зворотне тиск.

Для надійного зберігання даних було обрано популярну та надійну систему керування базами даних (далі СКБД) PostgreSQL.

4.1 Переваги обраних технологій розробки

Отже, виходячи з обраного напрямку розробки продукту, було обрано наступні технології:

1. мова програмування Scala;
2. фреймворк Akka;
3. мова програмування JavaScript;
4. програмна платформа Node.js;
5. бібліотека Puppeteer;

6. мова програмування Python;
7. бібліотека pytorch;
8. розподілений брокер повідомлень Apache Kafka;
9. СКБД PostgreSQL.

4.1.1 Scala

Scala поєднує об'єктно-орієнтоване та функціональне програмування однією стислою мовою високого рівня. Статичні типи Scala допомагають уникнути помилок у складних програмах, а її JVM та JavaScript під час виконання дозволяють створювати високопродуктивні системи з легким доступом до величезних екосистем бібліотек. [6]

Особливості Scala:

1. працює на JVM, тому стеки Java та Scala можна вільно змішувати для цілком безпроблемної інтеграції;

2. це чиста об'єктно-орієнтована мова програмування (ООР). Кожна змінна є об'єктом, і кожен “оператор” - це метод;

3. це також мова функціонального програмування (FP), тому функції також є змінними, і є можливість передавати їх іншим функціям. Можна написати код за допомогою ООР, FP або комбінувати їх у гібридному стилі;

4. не потрібно явно вказувати тип даних і тип значення функції. Тип значення, що повертається функції визначається типом останнього виразу, присутнього в функції;

5. ліниві обчислення за замовчуванням. Scala оцінює вирази лише тоді, коли вони потрібні. Існує можливість оголосити ліниву змінну за допомогою ключового слова lazy. Він використовується для підвищення продуктивності;

6. має гнучку систему типів та “домішок”;

7. має добре розроблені концепції обробки асинхронних задач. Ці концепції базуються на іммутабельності змінних, підходів, що забезпечують безпечний

доступ до конкурентних даних; Асинхронні обчислення організовані за рахунок абстракції Future, що відображає “ефект” дії на “задньому плані”.

4.1.2 Akka

Akka – це інструментарій та Виконавча для створення в JVM висококонкурентних, розподілених і відмовостійких подієво-орієнтованих додатків. Стек Akka - це безкоштовний інструментарій з відкритим вихідним кодом і середовище виконання, що спрощує створення паралельних і розподілених додатків на JVM. Akka підтримує кілька моделей програмування для паралелізму, але підкреслює паралелізм на основі акторів. [7]

Переваги Akka:

1. простий підхід до реалізації паралельних та розподілених систем. Абстракції Actor та Stream дозволяють створювати системи, які масштабуються, більш ефективно використовуючи ресурси сервера, і розширюються, використовуючи кілька серверів;
2. гнучкість. Грунтуючись на принципах «реактивного маніфесту», Akka дозволяє створювати системи, які самовиліковуються і залишаються стійкі до збоїв системи;
3. висока продуктивність. До 50 мільйонів повідомлень за секунду на одній машині. Невеликий обсяг займаємої оперативної пам'яті; ~ 2,5 мільйона акторів на один гігабайт купи;
4. еластичний і децентралізований. Розподілені системи без єдиної точки відмови. Балансування навантаження і адаптивна маршрутизація між вузлами. Event Sourcing і CQRS з кластерним поділом. Розподілені дані для можливої узгодженості з використанням CRDT;
5. реактивні потокові дані. Асинхронна неблокуюча обробка стрімінгових потоків даних із “зворотнім тиском” повідомлень при перевантажені. Повністю асинхронний і потоковий HTTP сервер та клієнт надають зручну платформу для побудови мікросервісів;

6. інтеграція з багатьма потоковими сервісами, у тому числі Apache Kafka.

4.1.3 Puppeteer

Puppeteer – це бібліотека Node, яка надає API високого рівня для управління Chrome або Chromium по протоколу DevTools. Puppeteer за замовчуванням працює у режимі headless (режим без відображення користувацького інтерфейсу, що дозволяє зекономити витрачаємі браузером ресурси), але може бути налаштований для запуску повного (без заголовка) Chrome або Chromium. [8]

Особливості бібліотеки Puppeteer:

1. надає доступ до вимірювання часу завантаження і рендеринга, що надається інструментом Chrome Performance Analysis;
2. надає більший контроль над браузерами Chrome, ніж інші бібліотеки (ймовірно, завдяки підтримці Google і глибоким знанням Chrome);
3. видаляє залежність від зовнішнього драйвера для запуску тестів;
4. за замовчуванням налаштований на роботу в режимі headless, і його також можна змінити, щоб спостерігати за ходом в реальному часі. [9]

4.1.4 JavaScript та Node.js

JavaScript (JS) – це невибаглива до ресурсів мова, що інтерпретується або точно скомпільований мову програмування з першокласними функціями. Хоча він найбільш відомий як мова сценаріїв для веб-сторінок, його також використовують багато не браузерних середовищ, такі як Node.js, Apache CouchDB і Adobe Acrobat. JavaScript - це мультипарадигменна, однопотокова, динамічна мова на основі прототипів, що підтримує об'єктно-орієнтовані, імперативні й декларативні стилі (наприклад, функціональне програмування). [10]

Node або Node.js – програмна платформа, заснована на движку V8 (здійснює трансляцію JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованої мови в мову загального призначення. Node.js додає

можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API (написаний на C ++), підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js і десктопні віконні додатки (за допомогою NW.js, AppJS або Electron для Linux, Windows і macOS) і навіть програмувати мікроконтролери (наприклад, tessel, low.js і espruino). В основі Node.js лежить подійно-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим введенням / виводом. [11]

Важливі для розробки сервісу збору зображень особливості Node.js:

1. пропонує легку масштабованість;
2. використовується в якості єдиної мови програмування як на клієнтській стороні, так і на серверній;
3. пропонує високу продуктивність;
4. середовище виконання Node.js з відкритим кодом також забезпечує можливість кешування окремих модулів. Щоразу, коли є запит на перший модуль, він зберігається в кеш-пам'яті програми;
5. надає можливість не блокувати системи вводу виводу, він порівняно допомагає вам одночасно обробляти кілька запитів;
6. має високу розширюваність, що означає, що розробник може налаштовувати і розширювати Node.js відповідно до своїх вимог.

4.1.5 Python та Pytorch

Python – високорівнева мова програмування загального призначення, орієнтований на підвищення продуктивності розробника і читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій. [12]

Python підтримує структурний, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване програмування. Основні архітектурні риси:

1. динамічна типізація;

2. автоматичне керування пам'яттю;
3. повна інтроспекція;
4. механізм обробки виключень;
5. підтримка багатопоточних обчислень;
6. високорівневі структури даних;
7. підтримується розбиття програм на модулі, які, в свою чергу, можуть об'єднуватися в пакети.

PyTorch – бібліотека машинного навчання для мови Python з відкритим вихідним кодом, створена на базі Torch. Використовується для вирішення різних завдань: комп'ютерний зір, обробка природної мови. Розробляється переважно групою штучного інтелекту Facebook. Також навколо цього фреймворка вибудувана екосистема, що складається з різних бібліотек, що розробляються сторонніми командами: Fast.ai, що спрощує процес навчання моделей, Pyro, модуль для імовірнісного програмування, від Uber, Flair, для обробки природної мови і Catalyst, для навчання DL і RL моделей. [13]

Особливості бібліотеки PyTorch:

1. заснований на Python
2. динамічний підхід до обчислення графів. PyTorch будує додатки для глибокого навчання на основі динамічних графів, з якими можна грати під час виконання. Інші популярні фреймворки глибокого навчання працюють над статичними графіками, де заздалегідь потрібно будувати обчислювальні графіки;
3. набагато простіше вивчити, ніж будь-яка іншу бібліотеку глибокого навчання, оскільки вона недалека від багатьох звичайних програмних практик;
4. має одну з найважливіших функцій, відомої як декларативний паралелізм даних. Ця функція дозволяє використовувати torch.nn.DataParallel для перенесення будь-якого модуля. Це буде розпаралелювання за розміром пакета, і ця функція допоможе вам легко використовувати кілька графічних процесорів.

4.1.6 Apache Kafka

Apache Kafka – це потокова платформа, яка має три ключові можливості:

1. публікація та підписка на потоки записів, подібні до черги повідомлень чи корпоративні системи обміну повідомленнями;
2. зберігає потоки записів у надійному та стійкому до відмов вигляді;
3. обробляє потоки записів по мірі їх виникнення. [14]

Kafka зазвичай використовується для двох широких класів застосувань: побудова потокових конвеєрів даних в реальному часі, які надійно отримують дані між системами або додатками; створення потокових програм у режимі реального часу, які перетворюють або реагують на потоки даних. [15]

Базові поняття:

1. Kafka запускається як кластер на одному або декількох серверах, які можуть охоплювати кілька центрів обробки даних.
2. кластер Kafka зберігає потоки записів у категоріях, що називаються «topic».
3. кожен запис складається з ключа, значення та часової позначки.

4.1.7 PostgreSQL

PostgreSQL – це система керування об'єктно-реляційними базами даних з відкритим вихідним кодом, яка використовує і розширює мову SQL в поєднанні з багатьма функціями, які безпечно зберігають та масштабують найскладніші робочі навантаження даних.

PostgreSQL має безліч розширених функцій, які пропонують інші системи управління базами даних, такі як:

4. користувач може визначати власні типи;
5. наслідування у таблицях;
6. вбудована підтримка слабоструктурованих даних в форматі JSON з можливістю їх індексації;

7. високопродуктивні і надійні механізми транзакцій і реплікації;
8. посилальна цілісність зовнішнього ключа;
9. вкладені транзакції (savepoints);
10. мультіверсіонность керування паралелізмом (MVCC);
11. асинхронна реплікація.

4.2 Висновки до розділу

У даному розділі було детально розглянуто наступні обрані технології їх переваги: мова програмування Scala, фреймворк Akka, мова програмування JavaScript, програмна платформа Node.js, бібліотека Puppeteer, мова програмування Python, бібліотека Pytorch, розподілений брокер повідомлень Apache Kafka, СКБД PostgreSQL. Було аргументовано використання тих чи інших підходів у зв’язці із зазначеними технологіями.

5 РОЗРОБКА СТРУКТУРИ СИСТЕМИ

5.1 Структура проєкту

Реалізація функціональних та нефункціональних вимог до проєкту вимагають ретельного проєктування архітектури. Саме тому для спрощення моделювання системи та досягнення гнучкості реалізації, було вирішено провести функціональну декомпозицію системи.

У системі можна виділити такі функціональні напрямки роботи сервісів:

1. пошук зображень та вилучення у вигляді посилань та деталей про зображення;
2. завантаження та вивантаження зображень;
3. співставлення зображень з певним класом;
4. точка взаємодії користувача з системою.

Спроєктувавши архітектуру системи, було виділено декілька окремих сервісів. На кресленику ІТ61.080БАК.004 Д2 наведено схему компонентів системи. Кожен являє собою окремий самостійний програмний продукт. Загалом, система буде складатись з чотирьох розроблених сервісів, одного стороннього та однієї системи керування базою даних.

Розроблені сервіси:

1. сервіс сбору зображень;
1. сервіс розпізнавання зображень;
2. шлюз програмного інтерфейсу системи;
3. сервіс завантаження зображень.

Сторонні сервіси:

1. брокер повідомлень Apache Kafka;
2. СКБД PostgreSQL.

Структура проєкта зображена на рисунку 5.1.

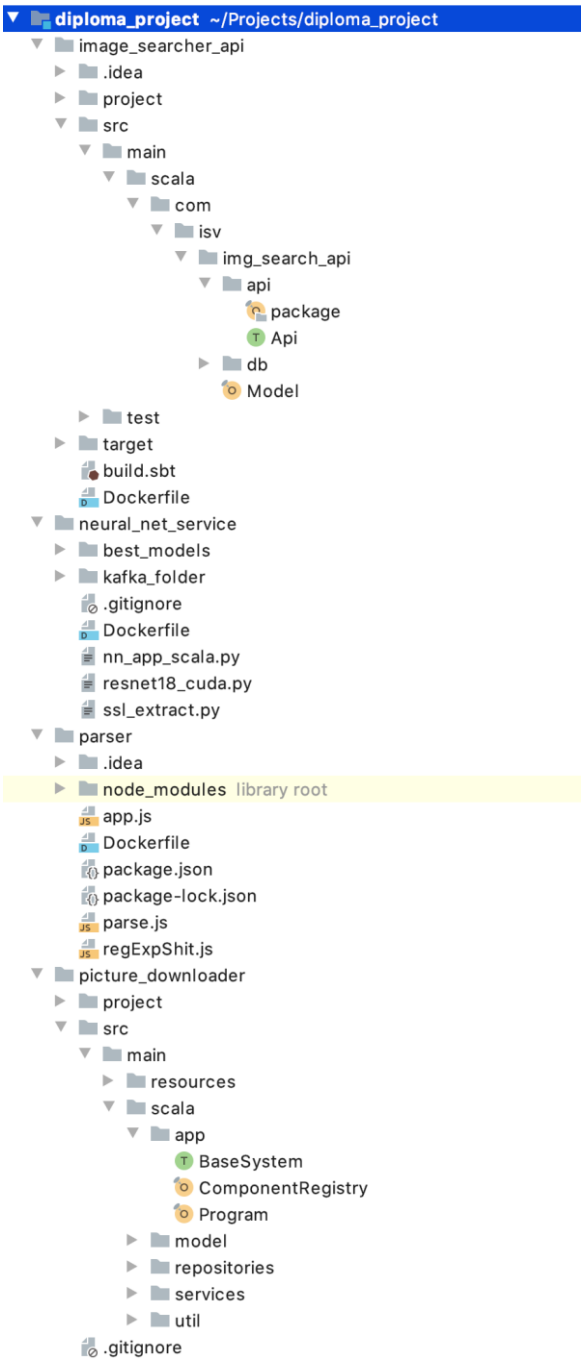

Рисунок 5.1 – структура проєкту

Взаємодію сервісів у випадку створення задачі на розпізнавання зображень за класами відображено на кресленику ІТ61.080БАК.004 Д3.

5.2 Детальний опис сервісів

Розглянемо детальніше окремі сервіси системи та їх роль у загальній архітектурі системи.

5.2.1 Сервіс збору зображень

Сервіс являє собою Node.js застосунок, який за допомогою драйверу керує певним браузером. У цьому випадку за робочий браузер було обрано Chromium. Для підвищення продуктивності роботи сервісу та зменшенню навантаження на ресурси середовища, у якому відбувається виконання програми, браузер запускається у полегшеному режимі, без графічного відображення. Керування програмним інтерфейсом браузеру необхідно для моделювання структурного дерева документу (DOM) та запуску необхідних скриптів як зі сторони клієнта (відрисовка сайту), так і зі сторони серверу (технічні скрипти для обробки інформації на сайті).

Сервіс сбору зображень працює за наступним сценарієм: сервіс очікує повідомлення, які надходять з брокеру повідомлень за визначеним протоколом. При отримані повідомлення початку нової задачі, сервіс починає переходити по вказаним посиланням за допомогою браузеру. Після того, як структура сайту згенерована, починається пошук зображень відповідно до налаштувань (якщо налаштування не були задані, то використовуються налаштування за замовчуванням). Посилання на знайдене зображення відправляється у інший сервіс системи – завантажувач. Процес відбувається доти, доки на кожному посиланні не закінчаться доступні зображення. По завершенню виконання задачі, відправляється повідомлення про успішне виконання конкретної задачі. Варто відзначити, що процес обробки задач відбувається асинхронно, тобто сервіс одночасно може обробляти багато таких задач.

5.2.2 Сервіс завантаження зображень

Даний сервіс являє собою Scala застосунок та призначений для завантаження медіаресурсів за вказаними посиланнями. На вхід сервіс отримує повідомлення з певним посиланням та додатковою інформацією.

Далі відбувається завантаження цих ресурсів за певним шляхом. Цей шлях визначається за певним правилом, в залежності від ідентифікатора задачі.

Окремим запитом створюється відповідний розділ у файловій системі середовища, куди сервіс буде зберігати завантажені ресурси у сгрупованому відповідно до ідентифікатора задачі вигляді.

Окрім функції завантаження сервіс також несе функцію вивантаження ресурсів. При створені задачі користувачем, API сервіс надсилає запит на цей сервіс про створення задачі. У відповідь повертається FTP посилання на каталог, у якому буде збережено результати пошуку зображень.

5.2.3 Сервіс розпізнавання зображень

Даний сервіс являє собою Python застосунок, що використовує сучасну бібліотеку Pytorch для використання натренированных моделей нейронних мереж. Які в свій час лежать в основі механізму розпізнавання зображень.

Найбільш простий, стандартною і при цьому досить поширеною проблемою машинного навчання є бінарна класифікація. До неї можна звести проблему відношення зображення до певного класу. Тому для обробки зображень на першому етапі було обрано технологію бінарних нейронних мереж.

Бінарні нейронні мережі – це мережі з бінарними вагами та активаціями під час виконання. Під час тренування ці ваги та активації використовуються для обчислення градієнтів; однак градієнти та справжні ваги зберігаються в повній точності. Ця процедура дозволяє ефективно навчати мережу в системах з меншою кількістю ресурсів.

Якщо нейромережі привласнюють кожному сегменту точно розраховану ймовірність, то бінарні нейронні мережі зводять ймовірні значення до чорно-білого варіанту, тобто, або до -1 (якщо мережа вважає, що ознаки в цьому фрагменті немає), або + 1 (якщо він є). Тобто зважена сума оцінює кожну ознаку або позитивно (множачи на +1), або негативно (множачи на -1), і замість повного

перемноження потрібно розглядати лише множення на +1 і -1. У двовимірному вигляді набір даних набуває вигляду зображеного на рисунку 5.2.

```
[1.2, 0.7] -> +1
[-0.3, 0.5] -> -1
[-3, -1] -> +1
[0.1, 1.0] -> -1
[3.0, 1.1] -> -1
[2.1, -3] -> +1
```

Рисунок 5.2 – приклад набору даних для бінарної нейронної мережі

Такий підхід жертвує певною часткою точності, але існує можливість відшкодувати втрати, збільшивши розмір мережі. Таким чином, двійкові мережі за своєю суттю набагато простіше.

У порівнянні зі своїми двійниками з плаваючою комою, їм потрібно в 32 рази менше місця для зберігання числа (1 біт замість 32), і в сотні разів менше енергії, через що вони більш застосовні в умовах обмежених ресурсів та підвищених навантажень. В таких умовах і буде працювати розроблювана система.

Ще однією перевагою використання бінарних нейронних мереж є можливість розширення кількості класів без необхідності перенавчання мультикласових нейронних мереж. Це відповідно надає можливість масштабувати кількість доступних для розгадування класів в режимі реального часу.

Перші треновані моделі були побудовані за допомогою ImageNet.

ImageNet – це проект зі створення і супроводу масивної бази даних анотованих зображень, призначена для відпрацювання і тестування методів розпізнавання образів і машинного зору.

Крім базових напрацювань із бази ImageNet, також було використано технологію ResNet.

Залишкова нейронна мережа (ResNet) - це штучна нейронна мережа, яка будується на конструкціях, відомих по пірамідальним клітинам в корі головного

мозку. Залишкові нейронні мережі роблять це, використовуючи припущення з'єднання або ярлики для переходу через деякі шари. Типові моделі ResNet реалізовані з використанням дво- або тришарових пропусків, які містять нелінійності (ReLU) і періодичну нормалізацію між ними. Пропуск шарів ефективно спрощує мережу, використовуючи меншу кількість рівнів на початкових етапах навчання. Це прискорює процес навчання, зменшуючи вплив зникаючих градієнтів, оскільки існує менше шарів, через які можна поширюватися.

Сценарій роботи сервісу: сервіс приймає на вхід повідомлення з брокера, де зазначені зображення та клас. Задача сервісу перевірити відповідність зображення класу. Якщо результат задовольняє налаштований процент схожості, то зображення рахується відповідним класу та зберігається у постійній пам'яті. В іншому випадку воно видаляється.

5.2.4 Шлюз програмного інтерфейсу системи

Даний сервіс є реалізацією паттерна Gateway на мікросервісному рівні. Суть такого підходу – це реалізація сервісу, який є точкою входу до програми із зовнішнього світу. Він несе відповідальність за маршрутизацію запитів, склад API та інші функції, такі як аутентифікація.

Тобто цей сервіс слугує вхідною точкою у розроблену систему. Він відкриває 80 або (при наявності SSL сертифікату) 443 порт для можливості встановлення HTTP з'єднання з клієнтом.

Основна функціональність сервісу відображає особливості функціональностей системи:

1. авторизація користувача;
2. створення, відображення, зупинка та перевірка стану задач;
3. додавання, редагування, видалення власних класів;
4. перегляд доступних класів у системі.

Сервіс написаний на язику Scala з використанням фреймворку Akka. Напряму відбувається взаємодія з базою даних для створення записів про зареєстрованих користувачів, наявні задачі та класи.

5.3 Розробка моделі зберігання даних

Сервіси, що взаємодіють з базою даних напряму, написані на стеку технологій Akka, Scala та Slick. Особливість фреймворку Slick полягає у тому, що для роботи з базами даних використовується вбудована функціонально-реляційна проекція (Functional–relational mapping). При такому підході, взаємодія з базою описана на мові Scala, а створення таблиць відповідних відношень у базі може бути виконано як з програмного коду, так і безпосередньо за допомогою мови визначення даних (DDL) у конкретній СКБД. У випадку цього проєкту – це PostgreSQL. Визначення моделі даних буде проводитися на стороні бази, розділяючи таким чином два шари архітектури: модель та бізнес-логіка.

Діаграма відношень сутностей у базі даних наведена на кресленику ІТ61.080БАК.004 Д4. Детальний опис призначення кожної розробленої таблиці зазначено у таблиці 5.1.

Таблиця 5.1 – опис призначення таблиць бази даних.

Назва відношення	Опис	Перелік атрибутів
user	Дані про користувача.	*uuid* – унікальний ідентифікатор користувача; *password* – секретний пароль користувача; *token* – ключ доступу до програмного інтерфейсу системи.
task	Дані про підзадачі на пошук зображень за певним посиланням.	*epic_uuid* – посилання на ідентифікатор агрегатора задач еріс. *url* – посилання на веб-сторінку, з якої необхідно зібрати зображення; *status* – статус виконання підзадачі.

Продовження таблиці 5.1

Epic	Дані про задачу на пошук зображень.	*uuid* – унікальний ідентифікатор задачі; *user_uuid* – посилання на ідентифікатор користувача, який спровокував виконання задачі; *status* – статус виконання задачі *regex* – регулярний вираз для спеціалізованого шляху пошуку картинок [опціональне]; *class_uuid* – посилання на ідентифікатор класу зображення; *container* – html тег, у середині якого потрібно шукати зображення [за замовчуванням – “body”] proxies – масив адрес проксі серверів.
Image	Дані про знайдене зображення.	*uuid* – ідентифікатор зображення; *url* – посилання на зображення в Інтернеті; *epic_uuid* – посилання на ідентифікатор агрегатора задач epic. *task_uuid* – ідентифікатор задачі, в процесі якої було завантажено зображення. *alt* – опис зображення відповідно до атрибуту alt тега img в структурі html сторінки;
image_class	Дані про клас зображень.	*uuid* – ідентифікатор класу зображень; *name* – ім’я класу у системі; *path* – шлях до файлу з тренованою нейромережою відповідного класу; *owner* – ідентифікатор власника класу (користувача або системи); *access* – режим доступу [public/private].

5.4 Розробка протоколу обміну даними

У даному розділі буде розглянуто опис протоколу обміну даними між сервісами там між клієнтом. Оскільки була обрана мікросервісна архітектура, то окрему увагу слід приділити розробці моделі об’єктів передачі даних. Ця модель повинна бути однаково описана у сервісах, що контактують між собою.

Основні шляхи обміну даними, які будуть використовуватись у системі – це TCP, FTP та HTTP.

FTP буде використовувати користувач для отримання доступу до результатів пошуку зображень.

HTTP з'єднання буде встановлюватись між деякими сервісами та клієнтом і сервісом, що надає програмний інтерфейс для користувача.

Для обміну даними між сервісами, а також для між користувачем та системою, головним MIME типом буде *application/json.*

JSON (JavaScript Object Notation) – простий формат обміну даними, зручний для читання і написання як людиною, так і комп'ютером. Він заснований на підмножині мови програмування JavaScript, визначеного в стандарті ECMA-262 3rd Edition. JSON - текстовий формат, повністю незалежний від мови реалізації. Ці властивості роблять JSON ідеальним мовою обміну даними. [16]

Це універсальна структура даних. Майже всі сучасні мови програмування підтримують їх в будь-якій формі. Логічно припустити, що формат даних, незалежний від мови програмування, повинен бути заснований на цих структурах.

Опишемо структуру запитів користувача до API системи відповідно до функціональних вимог системи у таблиці 5.2. Слід зазначити, що аутентифікація сесії відбувається за допомогою технології JWT (JSON Web Token).

Таблиця 5.2 – структура запитів до API системи.

URN	HTTP метод	Опис	Структура запиту
/auth	POST	Реєстрація користувача	Параметри тіла запиту: *email* – необхідний для ідентифікації користувача. *password* – секретний пароль користувача.
/public/classes	GET	Отримання списку усіх доступних класів у системі	

Продовження таблиці 5.2

/public/ classes/ {id}	GET	Отримання даних про конкретний клас.	Сегменти шляху: *id* – ідентифікатор конкретного класу.
/classes	GET	Отримання списку власних класів.	
/classes	POST	Створення нового класу у системі.	Параметри тіла запиту: *name* – унікальне для конкретного користувача ім’я класу. *access* – режим доступу. Може бути приватним або публічним. *file* – модель відповідного класу з розширенням .pt або .pth
	GET	Отримання конкретного класу за ідентифікатором	Сегменти шляху: *id* – ідентифікатор класу.
	PUT	Редагування класу за його ідентифікатором	Сегменти шляху: *id* – ідентифікатор класу. Параметри тіла запиту: name – нове ім’я класу *access* – новий режим доступу.
	DELETE	Видалення класу за його ідентифікатором	Сегменти шляху: *id* – ідентифікатор класу.
/tasks	POST	Створення нової задачі на пошук та завантаження зображень.	Параметри тіла запиту: *uris* – масив веб-посилань, з яких треба зібрати зображення. *regex* – регулярний вираз для спеціалізованого шляху пошуку картинок [опціональне]. *container* – html тег, у середині якого потрібно шукати зображення [за замовчуванням – “body”].

Продовження таблиці 5.2

/tasks	GET	Отримання списку усіх задач користувача.	
	GET	Отримання деталей конкретної задачі за ідентифікатором.	Сегменти шляху: *id* – ідентифікатор задачі
	DELETE		Сегменти шляху: *id* – ідентифікатор задачі

Опишемо протоколи обміну даними між сервісами системи у таблиці 5.3. Обмін буде відбуватися за допомогою брокера повідомлень Apache Kafka. Варто відзначити, що повідомлення для обміну також мають JSON формат.

Таблиця 5.3 – внутрішній протокол обміну даних.

Тема (topic)	Опис	Структура
parser_task	Подія створення задачі на запуск парсингу сайту за певним посиланням.	*user_uuid* – посилання на ідентифікатор користувача, що створив задачу; *epic_uuid* – посилання на ідентифікатор агрегатора задач епіс. *url* – посилання на веб-сторінку, з якої необхідно зібрати зображення; *status* – статус виконання підзадачі. image_class_uuid – клас розпізнавання зображення.
img_ref	Подія знайденого посилання на зображення на певній сторінці.	*user_uuid* – посилання на ідентифікатор користувача, що створив задачу; *epic_uuid* – посилання на ідентифікатор агрегатора задач епіс. *task_uuid* – посилання на ідентифікатор задачі (таблиця «task»); *ref* – посилання на зображення у Інтернеті.

Продовження таблиці 5.3

img_ unclassified	Подія завантаження зображення у систему.	*user_uuid* – посилання на ідентифікатор користувача, що створив задачу; *epic_uuid* – посилання на ідентифікатор агрегатора задач епіс. *task_uuid* – посилання на ідентифікатор задачі (таблиця «task»); *path* – шлях до каталогу з зображеннями; *image_class_uuid* – клас розпізнавання зображення; *image_uuid* – ідентифікатор зображення.
img_classified	Подія розпізнавання зображення за певним класом.	*user_uuid* – посилання на ідентифікатор користувача, що створив задачу; *epic_uuid* – посилання на ідентифікатор агрегатора задач епіс. *task_uuid* – посилання на ідентифікатор задачі (таблиця «task»); *image_uuid* – ідентифікатор зображення.

5.5 Висновки до розділу

У цьому розділі було описано архітектуру системи, реалізацію її окремих компонентів та зв’язок між ними. Кожний компонент був розглянутий відповідно до ролі у системі, а також до використаних технологій. Була описана модель обміну та збереження даних.

6 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Варто відзначити необхідність проведення тестування проєкту відповідно до наявних функціональних вимог та складності проєкту. Розповсюджена практика тестування програмного забезпечення – розробка автоматизованих тестів та паралельно проведення ручного тестування.

Оскільки розроблювана система базується на мікросервісній архітектурі, то її автоматизоване тестування буде поділятися на написання модульних тестів та інтеграційних. Перші дадуть змогу протестувати окремо різні функціональні особливості сервісу. Останні – інтеграцію окремих компонентів сервісу друг з другом.

6.1 Ручне тестування

Ручне тестування при розробці було використано у відношенні до декількох сервісів. Перш за все до сервісу збору зображень та до сервісу розпізнавання зображень. Тестування саме цих сервісів вручну обумовлено складністю написання автоматизованих тестів на основні випадки використання. Також вручну слід протестувати взаємодію між сервісами у функціонально важливих випадках використання.

Розглянемо результати ручного тестування сервісу збору зображень.

Головне функціональне призначення сервісу – збір зображень за вказаними посиланнями. Для тестування було обрано сайт, який є безкоштовним фотобанком ліцензійних зображень pixabay.com. Порядок дій для тестування цієї функціональності:

1. надсилання повідомлення про створення задачі на пошук зображень у брокер повідомлень;
2. спостереження за зображеннями, що додаються у відповідну до ідентифікатору задачі директорію;

3. порівняння отриманого набору посилань на зображення з тим, що повинне містити посилання на певну сторінку сайту;

4. перевірка наявності відповідного повідомлення про завершення виконання збору зображень з певного посилання у брокері повідомлень.

Розглянемо результати ручного тестування сервісу розпізнавання зображень.

Головне функціональне призначення сервісу – співставлення зображень з певним класом розпізнавання за допомогою нейронних мереж.

Оскільки в основі сервісу лежать нейронні мережі, то результат обчислення може достовірно оцінити лише людина. Саме тому цей сервіс був протестований вручну.

За допомогою брокера повідомлень, сервіс отримує тестові команди на розпізнавання тестових зображень. База зображень була завчасно підготовлена з різних Інтернет-джерел. Класифікація цих зображень була також проведена вручну, оскільки це єдиний спосіб прийнятно оцінити результати розпізнавання зображень.

Результати роботи сервісу при отриманні задачі на розпізнавання зображень відображено на рисунках 6.1 та 6.2.

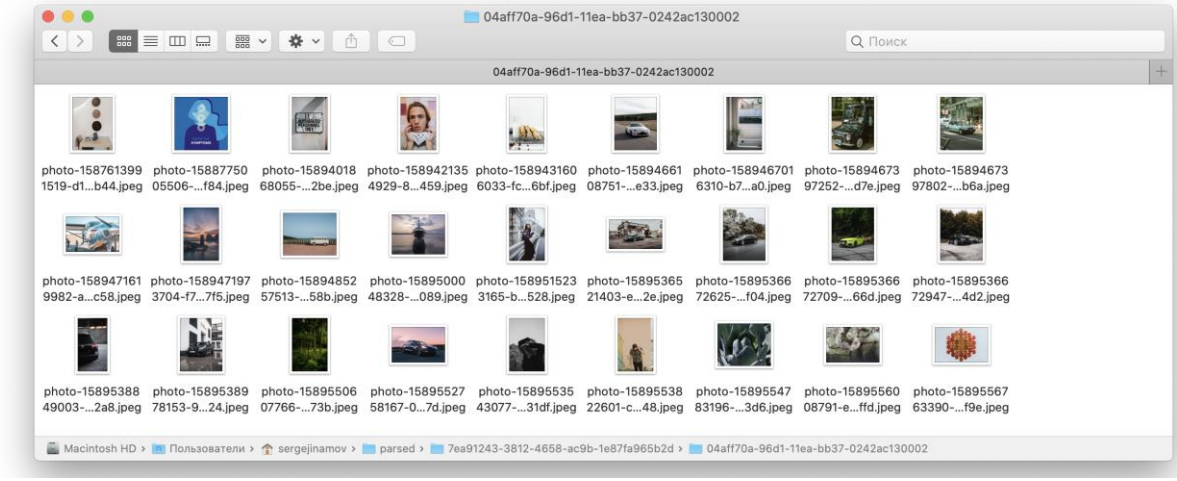

Рисунок 6.1 – приклад робочого каталогу із зображеннями

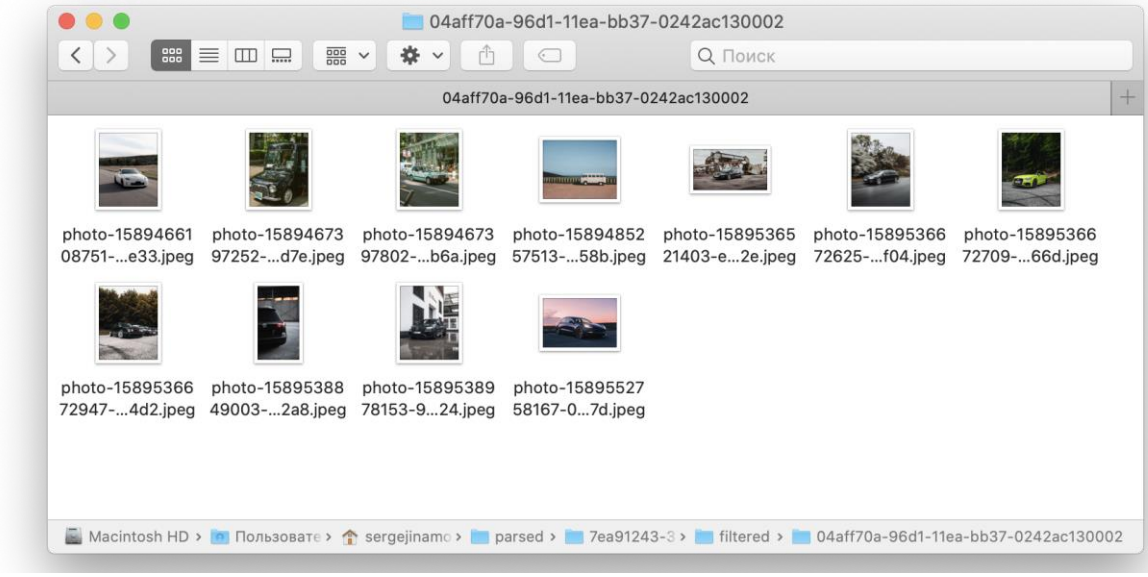

Рисунок 6.2 – приклад результату розпізнавання зображень за класом «Car»

6.2 Автоматизоване тестування

Суть автоматизованого тестування полягає у застосуванні програмних засобів для опису тестів, їх виконання та оформлення звітів щодо результатів тестування.

Для початку необхідно протестувати окремі компоненти системи та їх функціональність. Використаємо для цього підхід модульного тестування. Для цього необхідно спершу скласти тестові випадки використання сервісу. Опишемо окремо для кожного сервісу окремо такі випадки.

Тестові випадки для сервісу шлюзу програмного інтерфейсу зображено у таблицях 6.1, 6.2 та 6.3.

Таблиця 6.1 – Опис тестових випадків для інтерфейсу керування задачами.

Тестовий випадок	Очікуваний результат
Створення задачі	. статус код відповіді на запит 200; . задача створена у системі.

Продовження таблиці 6.1

Перевірка статусу активної задачі	. статус код відповіді на запит 200; 2. тіло відповіді на запит містить статус задачі «IN_PROGRESS» та ідентифікатор задачі.
Перевірка статусу скасованої задачі	. статус код відповіді на запит 200; 4. повернутий статус задачі «CANCELLED».
Перевірка статусу задачі, що очікує виконання	. статус код відповіді на запит 200; 6. повернутий статус задачі «PENDING».
Перевірка статусу завершеної задачі	. статус код відповіді на запит 200; 8. повернутий статус задачі «IN_PROGRESS».
Перевірка статусу завершеної задачі	. статус код відповіді на запит 200; 10. повернутий статус задачі «COMPLETE».
Перевірка статусу невиконаної задачі	1. статус код відповіді на запит 200; 2. повернутий статус задачі «FAILED».
Відображення історії задач	3. статус код відповіді на запит 200; 14. тіло відповіді на запит містить список задач, які закріплені за користувачем, що робить запит.
Відміна виконання задачі	5. статус код відповіді на запит 200; 16. поточний статус задачі «CANCELLED».

Таблиця 6.2 – Опис тестових випадків для інтерфейсу керування класами зображень.

Тестовий випадок	Очікуваний результат
Додавання нового класу	1. статус код відповіді на запит 200; 2. система завантажила та може використовувати новий клас.
Зміна налаштувань доступу до класу	1. статус код відповіді на запит 200; 2. доступ до класу змінено; 3. відповідь на запит до відображення доступних класів іншим користувачем змінився.
Відображення списку класів користувача.	1. статус код відповіді на запит 200; 2. тіло відповіді на запит містить список класів, які створив користувач.

Продовження таблиці 6.2

Видалення класу з системи	1. статус код відповіді на запит 200; 1. статус видалено з системи; 2. відповідь на запит до відображення доступних класів змінився.
Відображення списку усіх доступних класів.	1. статус код відповіді на запит 200; 2. тіло відповіді на запит містить список класів, які доступні користувачу.

Таблиця 6.3 – Опис тестових випадків реєстрації.

Тестовий випадок	Очікуваний результат
Реєстрація користувача з неіснуючою у системі email адресою.	1. статус код відповіді на запит 200; 2. користувач доданий до системи; 3. користувач отримав у тілі відповіді на запит унікальний ключ доступу до системи.
Реєстрація користувача з існуючою у системі email адресою.	1. статус код відповіді на запит 400; 2. користувач отримав помилку авторизації із вказаною причиною.

6.3 Інтеграційне тестування

Інтеграційне тестування – це етап тестування програмного забезпечення, де окремі сервіси поєднуються та перевіряються їх взаємодія в сукупності. Інтеграційне тестування є одним із підвидів автоматизованого тестування. Мета цього етапу тестування – виявити можливі несправності у взаємодії між інтегрованими сервісами. Інтеграційні тести відрізняється від інших тестових випадків тим, що вони зосереджується в основному на інтерфейсах та потоці даних та інформації між модулями. Тут слід надавати пріоритет інтеграційним посиланням, а не одиничним функціям, які вже перевірені. Саме тому, буде протестовано тільки декілька максимально повних сценаріїв використання системи. Тестові драйвери та тестові заглушки використовуються для допомоги в інтеграційному тестуванні.

При розробці системи було використано цей вид тестування враховуючи мікросервісну архітектуру, що лежить в основі, а також складність взаємодії між компонентами системи. Додаткову складність також створює використання різних

мов програмування, що використовувалися при розробці. Саме тому, інтеграційне тестування необхідне для контролю якості розроблюваної системи.

Кожний тестовий випадок – це окремий тестовий сценарій, де задіяні усі сервіси для виконання перевірки деякої функціональної можливості системи. Для більш швидкої розробки тестів для деяких компонентів системи були створені заглушки, а саме для сервісу пошуку зображень, а також для сервісу розпізнавання зображень. Описані сценарії, тестові випадки та їх результати наведені у таблицях 6.4 та 6.5.

Таблиця 6.4 – Тестовий випадок «Створення задачі на пошук зображень за певним класом, що визначений у системі»

Тестовий випадок	Створення задачі на пошук зображень за певним класом.
Очікуваний результат	1. статус код відповіді на запит створення нової задачі – 200; 3. задача створена у системі; 4. перевірка стану задачі повертає статус код відповіді на запит 200; 5. тіло відповіді на запит містить статус задачі «IN_PROGRESS» та ідентифікатор задачі; 6. через певний час така ж перевірка містить стан «COMPLETE»; 7. робочий каталог містить знайдені тестові зображення.
Результат	Тест пройдено.

Таблиця 6.5 – Тестовий випадок «Створення задачі на пошук зображень за класом, що створив користувач»

Тестовий випадок	Створення задачі на пошук зображень за класом, що створив користувач.

Продовження таблиці 6.5

Очікуваний результат	1. статус код відповіді на запит створення нового класу у системі – 200; 2. статус код відповіді на запит списку класів користувача – 200; 3. тіло відповіді на запит списку класів користувача містить створений тестовий клас; 4. статус код відповіді на запит створення нової задачі – 200; задача створена у системі; 9. перевірка стану задачі повертає статус код відповіді на запит 200; 10. тіло відповіді на запит містить статус задачі «IN_PROGRESS» та ідентифікатор задачі; 11. через певний час така ж перевірка містить стан «COMPLETE»; 12. робочий каталог містить знайдені тестові зображення.
Результат	Тест пройдено.

6.4 Висновок до розділу

У даному розділі було проведено огляд підходів до тестування розроблюваної системи. Були виділені та описані окремі тестові випадки та надано приблизні результати роботи окремих частин системи.

7 ВПРОВАДЖЕННЯ ТА ВИКОРИСТАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

7.1 Програмні та апаратні вимоги до експлуатації програми

Розроблювана система базується на мікросервісному архітектурному підході, тому окремі її частини (сервіси) мають різні апаратні та програмні вимоги.

Головна особливість вимог кожного такого сервісу – це навантаження на певну частину апаратних ресурсів. Розглянемо детальніше окремо вимоги до кожного сервісу.

7.1.1 Програмні та апаратні вимоги до сервісу збору зображень

Даний сервіс базується на технологіях Node.js, Puppeteer та Chromium. Головна точка навантаження – це процесор та оперативна пам’ять. Це зумовлено тим, що на кожну окрему задачу відкривається окремий процес браузеру і зростання навантаження буде зростати пропорціонально кількості задач у системі. Однак цей сервіс має можливості як горизонтального, так і вертикального масштабування, тому ефективність буде лише залежати від наявних ресурсів.

Опишемо мінімальні системні вимоги у випадку запуску не більше однієї задачі на сервер у таблиці 7.1 та рекомендовані системні вимоги у випадку виконання 100 задач одночасно.

Таблиця 7.1 – мінімальні системні вимоги сервісу збору зображень до апаратної частини сервера.

Процесор	Intel Pentium 4 / Athlon 64 або більш пізньої версії з підтримкою SSE2.
Місце на диску	700 Мб
Оперативна пам’ять	2 гігабайти.

Продовження таблиці 7.1

Операційна система	Windows 7 або пізніша, Mac OS X 10.6, Ubuntu 10.04 +, Debian 6 +, OpenSuSE 11.3 +, Fedora Linux 14

Таблиця 7.2 – рекомендовані системні вимоги сервісу збору зображень до апаратної частини сервера.

Процесор	Intel Xeon E-2244 або AMD аналог та більш пізні версії.
Місце на диску	700 Мб
Оперативна пам’ять	8 гігабайт.
Операційна система	Windows 7 або пізніша, Mac OS X 10.6, Ubuntu 10.04 +, Debian 6 +, OpenSuSE 11.3 +, Fedora Linux 14

7.1.2 Програмні та апаратні вимоги до сервісу завантаження зображень

Даний сервіс базується на технологіях Skala та Akka, не зберігає стану, а тому легко маштабується. Оскільки функціональне призначення полягає у завантажені зображень з інтернету та вивантаженні для користувача, то основне навантаження буде лягати на інтернет з’єднання та жорсткий диск. Сформуємо мінімальні та рекомендовані системні вимоги у таблицях 7.3 та 7.4 відповідно.

Таблиця 7.3 – мінімальні системні вимоги сервісу завантаження зображень.

Процесор	Intel Pentium 4 / Athlon 64 або більш пізньої версії з підтримкою SSE2.

Продовження таблиці 7.3

Місце на диску	125 Гб
Оперативна пам’ять	2 гігабайти
Операційна система	Windows 7 або пізніша, Mac OS X 10.6, Ubuntu 10.04 +, Debian 6 +, OpenSuSE 11.3 +, Fedora Linux 14,

Таблиця 7.4 – рекомендовані системні вимоги сервісу завантаження зображень.

Процесор	Intel Xeon Platinum 8259CL або більш пізньої версії.
Місце на диску	більше 1 терабайту
Оперативна пам’ять	8 гігабайт
Операційна система	Windows 7 або пізніша, Mac OS X 10.6, Ubuntu 10.04 +, Debian 6 +, OpenSuSE 11.3 +, Fedora Linux 14

7.1.3 Програмні та апаратні вимоги до сервісу розпізнавання зображень

Даний сервіс базується на технологіях Python та Pytorch, не зберігає стану, а тому легко маштабується. Оскільки функціональне призначення полягає у співставленні картинок відповідним класам за рахунок використання нейроних мереж, то основне навантаження буде лягати на процесор або графічний адаптер (відеокарту). Нейронні мережі побудовані за допомогою технологій Pytorh і можуть використовувати графічні адаптери, що підтримують архітектуру CUDA

(Compute Unified Device Architecture). Сформуємо мінімальні та рекомендовані системні вимоги у таблицях 7.5 та 7.6 відповідно.

Таблиця 7.5 – мінімальні системні вимоги сервісу завантаження зображень.

Процесор	Intel Pentium 4 / Athlon 64 або більш пізньої версії з підтримкою SSE2.
Графічний адаптер	GeForce GTX 660
Місце на диску	2 Гб
Оперативна пам’ять	2 гігабайти
Операційна система	Windows 7 або пізніша, Mac OS X 10.6, Ubuntu 10.04 +, Debian 6 +, OpenSuSE 11.3 +, Fedora Linux 14

Таблиця 7.6 – рекомендовані системні вимоги сервісу завантаження зображень.

Процесор	Intel Xeon Platinum 8259CL або більш пізньої версії.
Графічний адаптер	GeForce GTX 1080 Ti
Місце на диску	8 гігабайт і більше
Оперативна пам’ять	8 гігабайт
Операційна система	Windows 7 або пізніша, Mac OS X 10.6, Ubuntu 10.04 + Debian 6 + OpenSuSE 11.3 + Fedora Linux 14

7.1.4 Програмні та апаратні вимоги до шлюзу програмного інтерфейсу

Даний сервіс базується на технологіях Scala та Akka, не зберігає стану, а тому легко маштабується. Оскільки функціональне призначення полягає у обробці запитів користувачів до системи та взаємодія з базою, то основне навантаження буде лягати на процесор та оперативну пам’ять. Але так як сервіс не є високонавантаженим, то апаратні вимоги до нього відповідають вимогам технологій, що лежать в основі розробленого рішення. Сформуємо мінімальні та рекомендовані системні вимоги у таблицях 7.6 та 7.7 відповідно.

Таблиця 7.6 – мінімальні системні вимоги сервісу завантаження зображень.

Процесор	Intel Pentium 4 / Athlon 64 або більш пізньої версії з підтримкою SSE2.
Місце на диску	100 мегабайт
Оперативна пам’ять	512 мегабайт
Операційна система	Windows 7 або пізніша, Mac OS X 10.6, Ubuntu 10.04 + Debian 6 + OpenSuSE 11.3 + Fedora Linux 14

Таблиця 7.7 – рекомендовані системні вимоги сервісу завантаження зображень.

Процесор	Intel Xeon Platinum 8259CL або більш пізньої версії.
Місце на диску	100 мегабайт
Оперативна пам’ять	2 гігабайти
Операційна система	Windows 7 або пізніша, Mac OS X 10.6, Ubuntu 10.04 + Debian 6 +, OpenSuSE 11.3 +, Fedora Linux 14

7.2 Інструкція з встановлення

Описувана система має мікросервісну архітектуру, а тому є можливість встановлення її як окремими частинами на окремі сервери, так і усі компоненти загалом. Рекомендований підхід – встановлення окремих сервісів на окремі сервери або облачні платформи відповідно до описаних апаратних вимог кожного такого сервісу.

Для спрощення запуску окремих сервісів і не залежної від програмного оточення поведінки, було використано технологію Docker.

Docker – це набір продуктів «платформа як послуга» (PaaS), який використовує віртуалізацію на рівні ОС для доставки програмного забезпечення в пакетах, які називаються контейнерами. Контейнери ізольовані один від одного і включають в себе власне програмне забезпечення, бібліотеки і файли конфігурації; вони можуть спілкуватися один з одним за чітко визначеними каналами. Всі контейнери виконуються одним ядром операційної системи та рядом шарів, що описує розробник, і тому використовують менше ресурсів, ніж віртуальні машини.

Тому єдиною залежністю, яка необхідна для запуска кожного з сервісів буде встановлення Docker оточення останньої версії.

При умові встановлених залежностей, все що необхідно для запуска кожного компоненту системи – це виконати наступні термінальні команди, знаходячись у папці відповідного проєкту:

1. docker build -t {назва сервіу};
2. docker run {назва сервісу}.

Таким чином, буде запущено контейнер кожного сервіса з визначеним наперед програмним оточенням. Для кожного такого компоненту є можливість перевизначити параметри за замовчуванням, такі як:

1. {назва сервісу}_port (TCP порт, який буде відкрий для прослуховування зсередини контейнера);
2. {назва сервісу}_thread_count (кількість потоків процесора, які може створювати сервіс);

3. {назва сервісу}_task_count (кількість задач, які сервіс може обробляти паралельно);

4. {назва іншого сервісу}_host (адреса сервісу, від якого є залежність у форматі [ip:port]).

Ці параметри зчитуються сервісами зі змінних оточення, що дає можливість гнучко налаштовувати процес непреривного розгортання сервісів під час розробки.

Також присутня можливість неперервної інтеграції за рахунок використання інструментів платформи GitLab. Для цього необхідно мати піднятий сервер GitLab та GitLabRunner. Сценарії розгортання кожного сервісу описані у кожному каталозі кожного сервісу у відповідних файлах з назвою «.gitlab-ci.yml».

7.3 Висновки до розділу

У цьому розділі були описані системні та апаратні вимоги до кожного з компонентів розроблюваної системи. Обгрунтовано використання апаратних особливостей для кожного з сервісів. Також було наведено інструкцію з встановлення кожного компонента системи на сервер.

ВИСНОВКИ

У дипломному проєкті було розроблено автоматизовану систему пошуку медіафайлів за визначеними класами. Система дозволяє шукати в Інтернеті зображення за певними класами.

При виконані розробки було проведено аналіз предметної області розроблюваної системи. Було визначено призначення, цілі та задачі розробки, а також було визначено основні особливості майбутньої системи. Були проаналізовані існуючі рішення та створено детальний опис кожного з них. Були розглянуті та проаналізовані переваги та недоліки аналогів. На основі отриманих даних були розроблені функціональні та нефункціональні вимоги до системи. Далі, на основі вимог, визначених раніше, за допомогою обраних технологій було розроблено структуру системи та описані ключові тестові випадки.

Спроєктована система готова до тестування та впровадження. Функціональні вимоги дотримані. Існує можливість розширення функціоналу застосунку за допомогою модулів, які можна додати потім.

Проаналізувавши існуючі рішення, можна стверджувати, що розроблене програмне забезпечення має низку переваг, до яких можна віднести: унікальна функціональність, масштабованість, витримка високої нагрузки за умови відповідних обчислювальних можливостей, використання передових технологій.

Розробка автоматизованої системи пошуку медіафайлів за визначеними класами дозволила надати унікальну функціональність при порівняно невеликій складності системи. Можливість розширення робить систему відкритою для нової функціональності, а слабка зв'язність компонентів надає можливість для якісного покращення.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Semantic Web. URL: https://www.w3.org/standards/semanticweb (дата звернення: 10.05.2020).
2. Quicksprout. URL: https://www.quicksprout.com/the-advance-guide-to-seo-for-images (дата звернення: 12.05.2020).
3. Depositphoto. URL: https://ua.depositphotos.com (дата звернення: 15.05.2020).
4. REST API. URL: www.restfulapi.net (дата звернення: 19.05.2020).
5. Кун Роланд, Ханафи Брайан, Аллен Джейми. Реактивная архитектура сервисов. США, 2018. Кн. 1 С. 416 – 420.
6. Scala-lang. URL: www.scala-lang.org (дата звернення: 20.05.2020).
7. Akka. URL: https://akka.io (дата звернення: 21.05.2020).
8. Puppeteer. URL: https://pptr.dev (дата звернення: 23.05.2020).
9. Why-puppeteer-is-better-than-selenium. URL: https://www.lucidchart.com/techblog/2018/08/08/why-puppeteer-is-better-than-selenium (дата звернення: 25.05.2020).
10. JavaScript. URL: https://developer.mozilla.org/uk/docs/Web/JavaScript (дата звернення: 26.05.2020).
11. Node.js. URL: https://nodejs.org/uk (дата звернення: 15.05.2020).
12. Python https://www.python.org. URL: https://nodejs.org/uk/ (дата звернення: 19.05.2020).
13. PyTorch. URL: https://pytorch.org (дата звернення: 10.05.2020).
14. Apache Kafka. URL: https://kafka.apache.org (дата звернення: 9.05.2020).
15. Apache Kafka Uses. URL: https://kafka.apache.org/uses (дата звернення: 14.05.2020).
16. Json. URL: https://www.json.org/json-ru.html (дата звернення: 16.05.2020).